A semantic language for querying anonymous web sources

François Pinet, Michel Schneider

A Semantic Language for Querying Anonymous Web Sources

François Pinet[1] and Michel Schneider[1,2]

[1] Cemagref, 24 Avenue des Landais, 63172 Aubière Cedex, France
[2] LIMOS, Complexe des Cézeaux, 63173 Aubière Cedex, France
{francois.pinet, michel.schneider}@cemagref.fr

Abstract. A great deal of work has been carried out in recent years to facilitate access to data and information available on the Web. Proposals converge in two additional areas which consist in providing the sources with semantic annotations and in designing languages and tools that are capable of using these annotations. However, a large number of sources have not yet been annotated suitably. Besides, languages and existing tools do not allow the user to formulate "blind" queries without knowing the sources. To overcome these two limitations, in this paper we propose a flexible query language which allows a user to query sources in an anonymous way without knowing their existence and their structure. Queries can be solved by a system which in advance discovers potential sources and memorizes their schemas. We clarify how such a system can function.

1 Introduction

A large number of works in recent years have shown interest in the problem of retrieving data, documents, and knowledge available on the Web. The need to annotate available resources in order to facilitate their search is now well recognized. Complementary proposals have therefore been formulated, on one hand to annotate resources in computable semantic forms, and on the other hand to implement languages and systems capable of exploiting these annotations. Particular efforts have been devoted to the retrieval of knowledge and meta-data by benefiting from W3C standards. A synthesis of this trend can be found in [1]. It can also be observed that a certain number of works use these same standards for the retrieval of texts in documents [17]. Data retrieval from structured or semi-structured sources has also been subject to numerous investigations notably with a view to integration and mediation [3, 6, 15, 23]. The advantage of this type of approach is that it offers the user an integrated view of the data through a global schema. Its main drawback is the difficulty in maintaining the global schema when a source evolves or when a new source is integrated. Blind interrogation (without a view of a schema) of separate sources is an approach which has been investigated only to a lesser extent. Nevertheless such needs do exist. A simple example is the search for bibliographical information about works and authors among the different bibliographical sources available on the Web (e.g. what is the price of the book "Introduction to relational Databases", one author of

I. Lovrek, R.J. Howlett, and L.C. Jain (Eds.): KES 2008, Part II, LNAI 5178, pp. 106–116, 2008.

which is "Date"). It would be interesting to have a simple and structured language for querying these sources without knowing what their schema is (or even their existence), only from the concept names of the domain and from their relationships.

The works concerned by data searches in XML sources from keywords are quite close to our objective [2, 10, 14]. In these works it is a question of offering the user simple access without having to use Xpath or Xquery. Therefore an interface based on keywords is recommended. It is indeed a very simple device for the user but it introduces ambiguities in the specification of relationships (between concepts) and results. This problem of ambiguities is the source of interesting challenges for which satisfactory solutions are not yet completely available. To a certain extent these preoccupations are similar to those aiming to improve the efficiency of Web search engines [5, 7, 12, 13, 21, 22, 24].

We are convinced that it is possible to propose a simple and flexible language for the end user to achieve this kind of objective. The language SemanticSQL, which we present in this paper, interprets the intention of the user from the names of concepts and their relationships. It does not require the user to know the schemas of the sources or their existence. This language is qualified as flexible because it can adapt to badly annotated sources as well as richly annotated sources. We further show that a query in this language can be fairly matched with the schemas of relational sources or XML sources. We then sketch the functioning of a system capable of interpreting and resolving a user query. Such a system must discover the potential sources available on the Web in advance and memorize theirs schemas.

The paper is organized as follows. In section 2 we present the syntax of our language. In section 3 we give some indications on how the user and the system can co-operate. Section 4 defines the notion of valid query. In section 5 we explain the general matching process between a query and a potential source. Section 6 sets out details on how links matchings can be established. Section 7 provides comments on the results of certain experiments. Section 8 concludes and suggests some perspectives.

2 Syntax of the SemanticSQL Query Language

The syntax of our query language is organised in the same fashion as SQL. The query is divided up into three clauses with the reserved words "select", "from" and "where". As in SQL the "select" clause specifies the result of the query and the "where" clause specifies the condition which must be satisfied. The "from" clause has a different meaning. Since in our approach the user does not know the schemas of the sources, it cannot designate the structures from which the result must be extracted. So in the "from" clause the user indicates the names of types which he thinks might exist in the sources. An optional "on" clause makes it possible to restrict the query to certain sources.

The general form of a query is as follows:

Select list_of_variables **from** list_of_type_variable_pairs
[**where** list_of_conditions] [**on** list_of_sources]

list_of_variables: This is a list of variables introduced by the user. Names of variables are freely chosen by the user and are separated by commas. Names of variables are not related to the semantics of the query

type_variable_pair: This is a pair separated by a space composed of a type name and a variable name. It is used to allocate a variable to a type, so a variable marks an instance of a given type. Names of types in this clause are of great importance. They are directly related to the semantics of the query. Such pairs allow the user to specify the meaning of the objects which he is looking for.

list_of_type_variable_pairs: This is a list of type_variable_pairs separated by commas.

list_of_conditions: This is a list of conditions separated by commas where the comma has a particular significance: it marks the "and" operator. So the global condition of the where “clause” is true if each separate condition in this list is true.

condition := valuation_condition | link_condition

valuation_condition := elementary_condition |
(valuation_condition_1 or valuation_condition_2) |
(valuation_condition_1 and valuation_condition_2) |

elementary_condition := variable_name op constant | variable_name_1 op variable_name_2

op := > | < | >= | <= | <> | =

link_condition: := variable_name_1()variable_name_2 |
variable_name_1(role_name)variable_name_2 |
variable_name_1@variable_name_2

The first kind of conditions is used to constrain the values of a variable.

The second kind of conditions allows the user to specify the existence of links between query variables. This specification is made on a pair basis with the double symbol (). For example a()u means that the two variables a and u must be connected through some link. We can indicate the meaning of the link between the parentheses. For example a(write)b means that a must be connected to b through a link which means that "a writes b". The connection between a type and one of its attributes or data type properties is specified by the notation a@n.

The notation a()b specifies a non oriented link. The other two notations correspond to oriented links (from variable1 to variable2).

list_of_sources: This is a list of source names separated by commas.

The "where" and the "on" clauses are optional.

Example: Here is an example of a concrete query

Q1: Select n, t from book b, title t, author a, name n, university u where a(write)b, a@n, b@t, a()u, u='Stanford'

Query Q1 can be paraphrased as follows: search for the tuples [n, t] where n is an attribute which represents the name of an author a, t is an attribute which represents the title of a book b, a wrote b, a is linked to a university, the value of which is 'Stanford' (the semantics of the link does not matter).

Interpretation of a query: The interpretation of a query in our language is as follows: give all possible values for the list_of_variables which satisfy the condition of the where clause, each variable taking its values from among those of the type it represents.

3 Respective Responsibilities of the User and the System

In our approach we suppose that the user possesses some knowledge about the domain and its potential information (names of types, structures of these types). But it is not required that he knows the schema of the sources. In this section we give some indications on how the user and the system can cooperate in order to pose and to answer a query.

Source names
The user does not need to know the source names or the existence of certain sources. In such a case, the system should choose those that are the most suited to the user query from among the sources of its directory. Further along the user can indicate to the system sources concerned by its query.

Type names
The user chooses the most suitable names for each of the types concerned by his query. The choice of these type names is particularly important because it induces the interpretation of the query by the system. The user may be able to use an ontology of the domain known from the system. The system can widen its interpretation to synonyms and even to hyponyms of each of the names.

Link conditions
Even then, the user will fill in the links that he considers to be most plausible between the types which he has specified in the "from" clause. A named link should be preferred in order to facilitate the resolution of the query. However names of links (which correspond to names of properties or associations in the schemas of certain sources) can show considerable variety. This variety can make it difficult for the system to identify the sources which correspond best to the user's wish. Moreover links are not always named (as in the case of relational and XML sources). In certain cases it will be better to choose an anonymous link and to let the system identify the possibilities offered with sources. If the number of possibilities is too great, a dialogue with the user can make it possible to limit the search space. Usage of links such a@b can be restrictive. Certain sources (notably XML sources) can treat attributes as standard elements. So it will be necessary for the system to look for all the possible links which can exist between a and b even though the user indicates an attribute.

Valuation conditions
By specifying a valuation condition of the form "variable_name_1 op constant" the user supposes that type T associated to variable_1 is atomic and is compatible with the type of the constant. If for a given source, this type T is not atomic, a correct comparison cannot occur. So the system may search, from among the attributes or the descendants of this type, for some atomic type which can represent type T and whose value can be compared with the constant. This atomic type must be representative of type T. We can use the ontology to help in its identification. For example a non atomic type can be represented by an atomic type whose name is: "name", "label", …. The same problem arises for a valuation condition of the form "variable_name_1 op variable_name_2". For each type associated to the two variables, atomic descendants will need to be looked for whose values can be compared.

Form of the result
Depending on the sources, a type in the result can be atomic or structured. The display of the value of a structured type can create problems. It can be agreed that the value is the concatenation of the values of its atomic types. Alternatively, the system can display only the value of an element which identifies the result.

4 Valid Query

In order to be valid, a query has to respect certain constraints. We shall clarify these constraints by representing the query by a graph.

A query graph is constructed as follows. Each type is represented with a node which carries the name of the type. Each link between two variables is represented with an edge between the two associated types. A link a()b is represented with a non directed edge. A link a(role) b is represented with a directed edge from a towards b; this edge is labelled by role. A link a@b is represented with a directed edge in a dotted line from a towards b.

Definition (valid query): A query is valid if its graph is connected and with at most a single edge between each pair of nodes.

These constraints are justified as follows. First, if the graph is not connected, there are at least two separated components which each correspond to independent sub-queries. If there are two edges between the same pair of nodes, then there is a redundancy or incoherence in the meaning of the link which the user wishes to specify between the corresponding types.

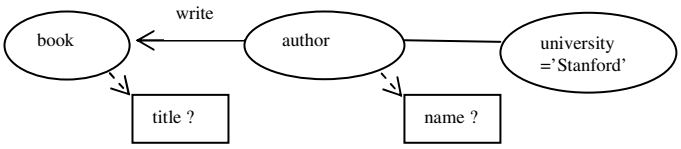

Fig. 1. Query graph of Q1

Query graph of Q1 is represented in figure 1 (the types are represented by ovals and the attributes by rectangles). This graph respects the two validity constraints and query Q1 is therefore valid.

5 General Principles for Matching a Query with Potential Sources

Answering the query means finding a correspondence between the query and each of the schemas of the potential sources.

Since a query can be represented by a graph, a solution for this problem consists in using a matching technique to establish this correspondence. Many matching algorithms have been suggested [4, 8, 16, 19, 20]. However these algorithms are difficult

to adjust. Besides, a query in our language is a very simple object compared to a schema and it would be interesting to study specific approaches for establishing a matching.

In this section we also provide a number of general indications on how the system can answer a query expressed with our language.

The matchinghas to consider two kinds of element: the names of types and the links between types. To obtain suitable answers for a query we suggest an approach consisting in first establishing the matching between the type names. It is only when a matching can be found for each type name that a source will be selected for the matching of links.

The matching of names can be very difficult if names are constructed freely by the users and the designers. We will suppose, in order to facilitate this stage, that the names of types in queries respect a domain ontology. So the ontology can provide a list of synonyms and a list of hyponyms for each type name of the query. The matching of a name with a source is then tested first with the name itself, then with its synonyms, then with its hyponyms. We do not require the names in the sources to respect the ontology. So names in sources are previously transformed in order to facilitate matching. Different kinds of string transformations have been proposed [4] and some of them are very efficient. For a given pair (a query, a schema), several possibilities of matching can exist for each query name. It is necessary to consider all the possibilities for the following stage of matching the links.

6 Some Guidelines for Matching the Links with XML Sources

In this section we provide a certain number of guidelines for matching links between a query and the schema of a XML source. Similar guidelines can be drawn with relational sources.

To illustrate these guidelines we consider the XML source of figure 2 which is a potential candidate for answering Q1.

The matching of type names gives the following correspondences:

title → title (of book)
name → name (of author), name (of grading_university, ofworking_university)
book → book
author → author
university → grading_university, working_university

So each type name for the query has at most one correspondence in the source. Subsequently we can try to match the links.

For the links book()title and author()name we can detect the correspondents easily (link between an element and one of its son for the first, link between an element and one of its attribute for the second). For the link author(write)book, it does not appear in the source a direct link between "author" and "book". But we observe that the word "writing" matches with "write" (the correspondence is established after lemmatization). So, we can compose the author→writing link (child-parent link) with the writing→book link (parent-child link) to establish a correspondence. Another problem is

that there are two links between "author" and "university" with two different meanings. In this case the system must propose the two possibilities and the user can then choose one of them. The last problem is that of the valuation condition university = "Stanford". If we consider the first correspondence (university→ grading_university), it is not possible to make the comparison directly. The system must detect that there is an atomic type which can represent the grading_university type and which permits the comparison. The attribute "name" can play this role. The automatic detection of such a possibility in all the possible situations is a difficult problem. The ontology of the domain can provide a certain number of indications. Otherwise, the system can display a partial schema of the source in order to ask for the user's opinion. For this valuation condition, another alternative is possible by considering the other correspondence university→ working_university. In this other case the comparison can be made using also the attribute "name".

A matching can thus be established for each link. The solution is not unique because there are two variants for the valuation condition.

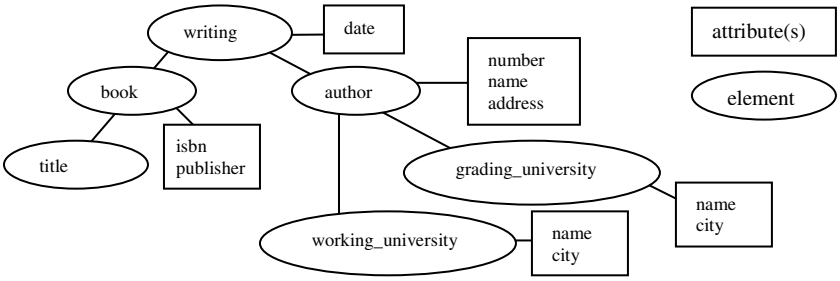

Fig. 2. XML tree for a source which matches with query Q1

In figure 3, we show the XML tree for another XML source with the same elements but organized differently. Although the link between book and author is not semantically characterized, the system must select this source for a matching with query Q1. The user will then confirm or not.

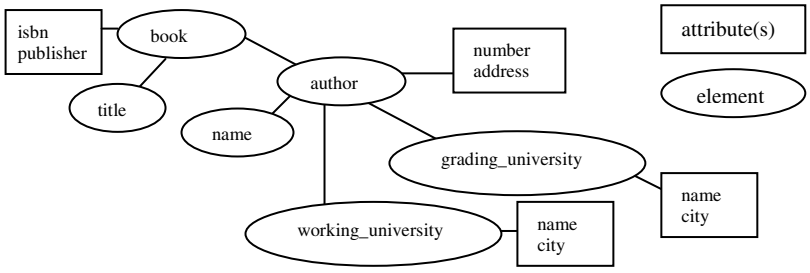

Fig. 3. XML tree for a second source which matches with query Q1

The XML source for which the element book would be a child of the element author (and not the parent as in figure 4) must also be selected by the system for the same query.

We formalize the previous analysis by the following rules.

R1: The link a()b of the query matches with a source S if A is a correspondent of a in the source, B is a correspondent of b and one of the following conditions holds:

i) it exists a direct link A,B or B,A in the source or B is an attribute of A or A is an attribute of B
ii) there exist a direct link A,C or C,A in the source and a path C,…,B where C and each intermediate node are in a relation of synonymy or hyperonymy or hyponymy with B
iii) same as ii) by exchanging the roles of A and B.

R2: The link a(c)b of the query matches with a source S if A, B, C are respectively correspondents of a, b, c in the source and one of the following conditions holds:

i) there exist a direct link A,C or C,A and a direct link C,B in the source or (symmetric case) there exist a direct link B,C or C,B and a direct link C,A
ii) there exist a path A,…,C or C,…,A in the source and a path C,…, B where each intermediate node is in a relation of synonymy or hyperonymy or hyponymy with A (resp. B)
iii) same as ii) by exchanging the roles of A and B.

R3: The link a@b of the query matches with a sources S if A is a correspondent of a in the source, B is a correspondent of b and one of the following conditions holds:

i) B is an attribute of A or a simple child of A
ii) it exists a path A,…,B in the source where each intermediate node is in a relation of synonymy or hyperonymy or hyponymy with A and B is a simple element
iii) it exists a path A,…,E in the source where each intermediate node and E are in a relation of synonymy or hyperonymy or hyponymy with A and B is an attribute of E.

R4: The condition a=v of the query matches with a source S if A is a correspondent of a in the source and one of the following conditions holds:

i) A is a simple element or an attribute (the condition is tested with A)
ii) it exists a path A,…,D in the source where each node is in a relation of synonymy or hyperonymy or hyponymy with A and D is a simple element (the condition is tested with D)
iii) it exists a path A,…,E where each intermediate node and E are in a relation of synonymy or hyperonymy or hyponymy with A and it exists an attribute of E having a name like ("name", "label", "description") (the condition is tested with this attribute.

Similar rules can be specified for relational sources.

7 Prototype and Experiments

To verify the ability of the language and the feasibility of our approach we have built a small prototype and conducted a number of experiments with XML sources. We used WORDNET [18] to help with the matching of names. Access to WORDNET was made through the JAVA API Java WordNet Library [11]. The body of the matcher was written in JAVA. For every link of the query, the various cases of matchings identified in section 5 are tested successively by considering all the possibilities offered with synonyms and hyponyms (at most level 3) for names. However matching of names is restricted to the domain (by using the ontology) to avoid meaning misunderstanding. We have only tested queries with links of the type a()b and conditions.

In the first stage, our experiments were conducted on six different XML sources containing data on sales of products. Sources were built manually. They contained from 8 to 14 elements. Every element had on average two attributes. We submitted ten different queries to the system. The matchings performed quite well. 90% of the total returned matchings were correct and 85% of the total correct matchings were retrieved. The incorrect matchings resulted essentially from a bad detection of the type replacement in the valuation conditions.

In the second stage we implemented a module in our prototype to discover potential sources in the same domain (sales of product) on the Web and to memorize their DTD. After validation by the administrator, 10 different sources were then incorporated into the system. We submitted the same 10 queries and we observed that the matchings performed badly. It appears in fact that several element names in the DTD were abbreviations which cannot be handled correctly by our string transformations. So we decided to create a specific dictionary for the management of these abbreviations and we significantly increased the efficiency of the matches. 80% of the returned matchings were correct. Bad type replacements partially explained incorrect matchings. Another cause of error was observed: a nonsense in the matching of names. These sources were rather complex and it was difficult to list manually all the correct matchings for each query. We estimated that the prototype had discovered about 75 % of them.

From these experiments it appears that the approach is realistic. The main points which condition its efficiency are the type replacement in valuation conditions and the detection of the meaning of names.

8 Conclusion and Perspectives

In this paper we proposed a query language for a final user which allows blind accesses to Web data sources. The user formulates his query by specifying the names of types and relationships between these types. It is not necessary for the user to know the existence of sources or their schemas.

This language is flexible. It can adapt to sources whether well annotated or not.

We discussed how a system can analyze a query and elaborate the results. Such a system must discover the potential sources in advance (for the considered domain) and memorize their schemas. To answer a query it first has to look for the matchings of names and then for the matchings of links. It proposes all the solutions and the user

validates those with which he is interested. The system can then rewrite the query for the corresponding sources.

We conducted experiments with XML sources which establish the efficiency of the language and the feasibility of the approach. Matchings can be improved on two points: type replacement in the valuation conditions and detection of the meaning of names. Concerning this last point one can reduce the ambiguities by working in a precise domain. It is possible also to take into account the profile of the user which can be acquired through a dialogue or by observing the queries. One can also observe how the user validates the solutions proposed by the system. Another way is to group results according to their different meanings [9].

Others improvements can be envisaged. One could exploit the mappings which can exist between sources to permit joins between these sources. The graph of a query is very simple and expressive and it would be interesting to study how it can support a graphical query interface. To improve the matching of links one also can ask the user to indicate the cardinalities.

Such a system can be very useful for different applications. Incorporated into an intranet system, it would allow a user to reach the data sources without knowing their schemas, by being based only on the domain ontology. In a P2P system, it could be installed on some peers or on super-peers to facilitate access to data by their semantics.

References

1. Bailey, J., Bry, F., Furche, T., Schaffert, S.: Web and Semantic Web Query Languages: A Survey. Reasoning Web, 35–133 (2005)
2. Cohen, S., Mamou, J., Kanza, Y., Sagiv, Y.: XSEarch: A Semantic Search Engine for XML. In: VLDB 2003, pp. 45–56 (2003)
3. Cui, Z., Jones, D., O'Brien, P.: Issues in Ontology-based Information Integration. In: IJCAI, Seattle (2001)
4. Do, H.H., Rahm, E.: COMA - A System for Flexible Combination of Schema Matching Approaches. In: VLDB 2002, pp. 610–621 (2002)
5. Duke, A., Glover, T., Davies, J.: Squirrel: An Advanced Semantic Search and Browse Facility. In: Franconi, E., Kifer, M., May, W. (eds.) ESWC 2007. LNCS, vol. 4519, pp. 341–355. Springer, Heidelberg (2007)
6. Garcia-Molina, H., Papakonstantinou, Y., Quass, D., Rajaraman, A., Sagiv, Y., Ullman, J., Vassalos, V., Widom, J.: The Tsimmis approach to mediation: Data models and languages. Journal of Intelligent Information Systems 8(2), 117–132 (1997)
7. Goldschmidt, D.E., Krishnamoorthy, M.S.: Comparing keyword search to semantic search: a case study in solving crossword puzzles using the Google™ API. Softw. Pract. Exper. (SPE) 38(4), 417–445 (2008)
8. Hai Do, H., Melnik, S., Rahm, E.: Comparison of Schema Matching Evaluations. Web, Web Services, and Database Systems, pp. 221–237 (2002)
9. Hemayati, R., Meng, W., Yu, C.T.: Semantic-Based Grouping of Search Engine Results Using WordNet. In: Dong, G., Lin, X., Wang, W., Yang, Y., Yu, J.X. (eds.) APWeb/WAIM 2007. LNCS, vol. 4505, pp. 678–686. Springer, Heidelberg (2007)
10. Hristidis, V., Koudas, N., Papakonstantinou, Y., Srivastava, D.: Keyword Proximity Search in XML Trees. IEEE Trans. Knowl. Data Eng (TKDE) 18(4), 525–539 (2006)

11. JWNL. Java WordNet Library, http://sourceforge.net/projects/jwordnet
12. Kandogan, E., Krishnamurthy, R., Raghavan, S., Vaithyanathan, S., Zhu, H.: Avatar semantic search: a database approach to information retrieval. In: SIGMOD 2006, pp. 790–792 (2006)
13. Li, Y., Wang, Y., Huang, X.: A Relation-Based Search Engine in Semantic Web. IEEE Trans. Knowl. Data Eng (TKDE) 19(2), 273–282 (2007)
14. Liu, Z., Walker, J., Chen, Y.: XSeek: A Semantic XML Search Engine Using Keywords. In: VLDB 2007, pp. 1330–1333 (2007)
15. Lenzerini, M.: Logical Foundations for Data Integration. In: Vojtáš, P., Bieliková, M., Charron-Bost, B., Sýkora, O. (eds.) SOFSEM 2005. LNCS, vol. 3381, pp. 38–40. Springer, Heidelberg (2005)
16. Madhavan, J., Bernstein, P.A., Rahm, R.: Generic Schema Matching with Cupid. In: VLDB 2001, pp. 49–58 (2001)
17. Mangold, C.: A survey and classification of semantic search approaches. IJMSO 2(1), 23–34 (2007)
18. Miller, G.: Wordnet: A Lexical Database for English. Communications of the ACM 38, 39–41 (1995)
19. Mohsenzadeh, M., Shams, F.: Teshnehlab M.: Comparison of Schema Matching Systems. WEC (2), 141–147 (2005)
20. Rahm, E., Bernstein, P.A.: A survey of approaches to automatic schema matching. VLDB Journal 10(4), 334–350 (2001)
21. Royo, J.A., Mena, E., Bernad, J., Illarramendi, A.: Searching the Web: From Keywords to Semantic Queries. In: ICITA 2005, pp. 244–249 (2005)
22. Tran, T., Cimiano, P., Rudolph, S., Studer, R.: Ontology-Based Interpretation of Keywords for Semantic Search. In: Aberer, K., Choi, K.-S., Noy, N., Allemang, D., Lee, K.-I., Nixon, L., Golbeck, J., Mika, P., Maynard, D., Mizoguchi, R., Schreiber, G., Cudré-Mauroux, P. (eds.) ASWC 2007 and ISWC 2007. LNCS, vol. 4825, pp. 523–536. Springer, Heidelberg (2007)
23. Wiederhold, G.: Mediators in the architecture of future information systems. IEEE Computer 25(3), 38–49 (1992)
24. Zhou, Q., Wang, C., Xiong, M., Wang, H., Yu, Y.: SPARK: Adapting Keyword Query to Semantic Search. In: Aberer, K., Choi, K.-S., Noy, N., Allemang, D., Lee, K.-I., Nixon, L., Golbeck, J., Mika, P., Maynard, D., Mizoguchi, R., Schreiber, G., Cudré-Mauroux, P. (eds.) ASWC 2007 and ISWC 2007. LNCS, vol. 4825, pp. 694–707. Springer, Heidelberg (2007)